

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier      : Famile.cpp  
5 Auteur(s)    : Gabrielli Alexandre , Povoà Tiago  
6 Date        : 22.05.2018  
7  
8 But         : implemente les fonctions de la class Famile  
9 -----  
10 */  
11 #include "Famile.h"  
12 #include "Voleur.h"  
13 #include "Policier.h"  
14  
15 Famile::Famile(const std::string &name) : Person(name) {}  
16  
17 Famile::Famile(const std::string &name, bool drive) : Person(name, drive) {}  
18
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Famile.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : la class Famille represente une famille, toute classe héritante  
9 appartient à la famille.  
10 -----  
11 */  
12 #ifndef POO2_LABO4_FAMILLE_H  
13 #define POO2_LABO4_FAMILLE_H  
14  
15 #include "Person.h"  
16  
17  
18 class Famille : public Person {  
19     virtual std::string type_info() = 0;  
20  
21 protected:  
22     explicit Famille(const std::string &name);  
23     explicit Famille(const std::string &name, bool drive);  
24  
25 };  
26  
27  
28  
29  
30 #endif //POO2_LABO4_FAMILLE_H  
31
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier      : Fille.cpp  
5 Auteur(s)    : Gabrielli Alexandre , Povoà Tiago  
6 Date        : 22.05.2018  
7  
8 But         : implemente les fonctions de la class Fille  
9 -----  
10 */  
11  
12 #include "Fille.h"  
13 #include "Papa.h"  
14 #include "Mamam.h"  
15  
16 Fille::Fille(const std::string &name) : Famile(name,false) {  
17 }  
18  
19  
20 std::string Fille::type_info() {  
21     return typeid(Fille).name();  
22 }  
23
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Fille.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : la class Fille represente une fille est hérite de famille , toutes  
9 filles ne peut pas conduire  
10 -----  
11 */  
12  
13 #ifndef POO2_LABO4_FILLE_H  
14 #define POO2_LABO4_FILLE_H  
15  
16  
17 #include "Famile.h"  
18  
19 class Fille : public Famile {  
20  
21  
22 public:  
23     std::string type_info() override;  
24  
25     explicit Fille(const std::string &name);  
26 };  
27  
28  
29 #endif //POO2_LABO4_FILLE_H  
30
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier      : Fils.cpp  
5 Auteur(s)    : Gabrielli Alexandre , Povoà Tiago  
6 Date        : 22.05.2018  
7  
8 But          : implemente les fonctions de la class Fils  
9 -----  
10 */  
11 #include "Fils.h"  
12  
13 Fils::Fils(const std::string &name) : Famille(name, false) {  
14 }  
15  
16 std::string Fils::type_info() {  
17     return typeid(Fils).name();  
18 }  
19  
20
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Fils.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : la class Fils represente un fils est hérite de famille ,tous  
9 fils ne peux pas conduire  
10 -----  
11 */  
12  
13 #ifndef POO2_LABO4_FILS_H  
14 #define POO2_LABO4_FILS_H  
15  
16  
17 #include "Famile.h"  
18  
19 class Fils : public Famile {  
20  
21     std::string type_info() override;  
22 public:  
23     explicit Fils(const std::string& name);  
24 };  
25  
26  
27 #endif //POO2_LABO4_FILS_H  
28
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier      : Mamam.cpp  
5 Auteur(s)    : Gabrielli Alexandre , Povoà Tiago  
6 Date        : 22.05.2018  
7  
8 But          : implemente les fonctions de la class Mamam  
9 -----  
10 */  
11  
12 #include "Mamam.h"  
13  
14  
15 Mamam::Mamam(std::string string): Famile(string) {  
16  
17 }  
18  
19 std::string Mamam::type_info() {  
20     return typeid(Mamam).name();  
21 }  
22  
23
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Mamam.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : la class Mamam represente une mamam est hérite de Famille, les  
9 mamams peuvent conduire  
10 -----  
11 */  
12 #ifndef POO2_LABO4_MAMAM_H  
13 #define POO2_LABO4_MAMAM_H  
14  
15  
16 #include "Famile.h"  
17  
18 class Mamam : public Famile {  
19  
20 public:  
21     std::string type_info() override;  
22  
23     explicit Mamam(std::string string);  
24 };  
25  
26  
27 #endif //POO2_LABO4_MAMAM_H  
28
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Papa.cpp  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : implemente les fonctions de la class Papa  
9 -----  
10 */  
11  
12 #include "Papa.h"  
13  
14  
15 Papa::Papa(std::string string) :Famile(string){  
16  
17 }  
18  
19 std::string Papa::type_info() {  
20     return typeid(Papa).name();  
21 }  
22
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Papa.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : la class papa represente un père est hérite de famille, les  
9 pères peuvent conduire  
10 -----  
11 */  
12 #ifndef POO2_LABO4_PAPA_H  
13 #define POO2_LABO4_PAPA_H  
14  
15  
16 #include "Famile.h"  
17  
18 class Papa : public Famile {  
19  
20  
21 public:  
22     std::string type_info() override;  
23  
24     explicit Papa(std::string string);  
25 };  
26  
27  
28 #endif //POO2_LABO4_PAPA_H  
29
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Person.cpp  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : implemente les méthodes de Person. On peut récupérer leur nom, et  
9 savoir si ils peuvent conduire le bateau  
10 -----  
11 */  
12  
13 #include "Person.h"  
14  
15  
16 std::ostream &operator<<(std::ostream &os, const Person &person) {  
17     os << person.name;  
18     return os;  
19 }  
20  
21 Person::Person(const std::string &name) : name(name), drive(true) {}  
22  
23 Person::Person(const std::string &name, bool drive) : name(name), drive(drive) {}  
24  
25 const std::string &Person::getName() const {  
26     return name;  
27 }  
28  
29 bool Person::canDrive() {  
30     return drive;  
31 }  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Person.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : Déclare les méthodes de nos classes Person, Voleur, Policier, Normal,  
9 Adulte et enfant. Chaque personne a un nom et peut conduire ou non  
10 -----  
11 */  
12  
13  
14 #ifndef POO2_LABO4_PERSON_H  
15 #define POO2_LABO4_PERSON_H  
16  
17 #include <iostream>  
18 #include <map>  
19 #include <memory>  
20 #include "string"  
21  
22  
23 class Person {  
24  
25     std::string name;  
26     bool drive;  
27  
28 public:  
29     bool canDrive();  
30  
31     virtual std::string type_info() = 0;  
32  
33     explicit Person(const std::string &name);  
34     explicit Person(const std::string &name, bool drive);  
35  
36  
37     const std::string &getName() const;  
38  
39     friend std::ostream &operator<<(std::ostream &os, const Person &person);  
40 };  
41  
42  
43 #endif //POO2_LABO4_PERSON_H  
44
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier      : Policier.cpp  
5 Auteur(s)    : Gabrielli Alexandre , Povoà Tiago  
6 Date        : 22.05.2018  
7  
8 But          : implemente les fonctions de la class Policier  
9 -----  
10 */  
11  
12 #include "Policier.h"  
13  
14  
15  
16 Policier::Policier(std::string string) : Person(string) {}  
17  
18 std::string Policier::type_info() {  
19     return typeid(Policier).name();  
20 }  
21
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Policier.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : la class Policier represente un policier est hérite de Personne,  
9 un policier peu conduire  
10 -----  
11 */  
12  
13 #ifndef POO2_LABO4_POLICIER_H  
14 #define POO2_LABO4_POLICIER_H  
15  
16  
17 #include "Person.h"  
18  
19 class Policier : public Person {  
20     std::string type_info() override;  
22 public:  
23     explicit Policier(std::string string);  
24 };  
25  
26  
27 #endif //POO2_LABO4_POLICIER_H  
28
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier     : Voleur.cpp  
5 Auteur(s)   : Gabrielli Alexandre , Povoà Tiago  
6 Date        : 22.05.2018  
7  
8 But         : implemente les fonctions de la class Voleur  
9 -----  
10 */  
11  
12 #include "Voleur.h"  
13  
14  
15 Voleur::Voleur(std::string string) : Person(string, false) {}  
16  
17 std::string Voleur::type_info() {  
18     return typeid(Voleur).name();  
19 }  
20
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Voleur.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : la class Voleur represente un policier est hérite de Personne,  
9 un policier ne peu pas conduire  
10 -----  
11 */  
12  
13 #ifndef POO2_LABO4_VOLEUR_H  
14 #define POO2_LABO4_VOLEUR_H  
15  
16  
17 #include "Person.h"  
18  
19 class Voleur : public Person {  
20     std::string type_info() override;  
21 public:  
22     explicit Voleur(std::string string);  
23 };  
24  
25  
26 #endif //POO2_LABO4_VOLEUR_H  
27
```

```

1  /*
2  -----
3  Laboratoire : 04
4  Fichier      : AbstactContainer.cpp
5  Auteur(s)    : Gabrielli Alexandre , Povoà Tiago
6  Date        : 22.05.2018
7
8  But          : implémente les méthode de la classe AbstactContainer.
9  -----
10 */
11
12 #include <utility>
13 #include <cstring>
14 #include <iostream>
15 #include <Person/Voleur.h>
16 #include <Person/Policier.h>
17 #include <Person/Mamam.h>
18 #include <Person/Fille.h>
19 #include "AbstractContainer.h"
20
21 void AbstractContainer::addMember(Person *person) {
22     this->persons.push_back(person);
23 }
24
25
26 AbstractContainer::AbstractContainer(const std::string &name) : name(name) {}
27
28 std::ostream &operator<<(std::ostream &os, const AbstractContainer &container) {
29     return os << container.toString();
30 }
31
32
33 bool AbstractContainer::removeMember(const Person *person) {
34     for (auto it = persons.begin(); it != persons.end(); ++it) {
35         if (((Person *) *it)->getName() == person->getName()) {
36             persons.erase(it);
37             return true;
38         }
39     }
40     return false;
41 }
42
43 bool AbstractContainer::verifie() {
44     bool isBoyPresent = false;
45     bool isGirlPresent = false;
46     bool isFatherPresent = false;
47     bool isMotherPresent = false;
48     bool isThiefPresent = false;
49     bool isCopPresent = false;
50
51     for (const auto &person : persons) {
52         if (person->type_info() == typeid(Voleur).name()) {
53             isThiefPresent = true;
54         } else if (person->type_info() == typeid(Policier).name()) {
55             isCopPresent = true;
56         } else {
57             if (person->canDrive()) {
58
59                 if (person->type_info() == typeid(Mamam).name()) {
60                     isMotherPresent = true;
61                 } else {
62                     isFatherPresent = true;
63                 }
64             } else {
65                 if (person->type_info() == typeid(Fille).name()) {
66                     isGirlPresent = true;
67                 } else {
68                     isBoyPresent = true;
69                 }
70             }
71         }
72     }
73 }
```

```
73     }
74
75 // vérification des contraintes
76 // policier - voleur
77     if (isThiefPresent && !isCopPresent && (isFatherPresent || isMotherPresent || isBoyPresent || isGirlPresent)) {
78         std::cout <<
79             "# Le voleur ne peut rester en contact avec un membre de la famille si le policier n'est pas present !"
80             << std::endl;
81         return false;
82     }
83 // mère - fils
84     if (isMotherPresent && isBoyPresent && !isFatherPresent) {
85         std::cout << "# Les fils ne peuvent rester seuls avec leur mere si le pere n'est pas present !"
86             << std::endl;
87         return false;
88
89     }
90
91 // père - fille
92     if (isFatherPresent && isGirlPresent && !isMotherPresent) {
93         std::cout << "# Les filles ne peuvent rester seules avec leur pere si la mere n'est pas presente !
94             "
95             << std::endl;
96         return false;
97     }
98     return true;
99 }
100 const std::list<Person *> &AbstractContainer::getPersons() const {
101     return persons;
102 }
103
104 const std::string &AbstractContainer::getName() const {
105     return name;
106 }
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier     : Bank.cpp  
5 Auteur(s)   : Gabrielli Alexandre , Povoà Tiago  
6 Date        : 22.05.2019  
7  
8 But         : implémente les méthodes de la classe Bank.  
9 -----  
10 */  
11  
12  
13 #include "Bank.h"  
14  
15  
16 std::string Bank::toString() const {  
17     std::string string;  
18     string = this->name + " : ";  
19     for (Person *person : this->persons) {  
20         string += person->getName() + ' ';  
21     }  
22     return string;  
23 }  
24  
25 }  
26  
27 Bank::Bank(const std::string &name, std::initializer_list<Person *> persons) : AbstractContainer(name) {  
28     for (Person *person : persons) {  
29         this->addMember(person);  
30     }  
31 }  
32  
33 Bank::Bank(const std::string &name) : AbstractContainer(name) {}
```

```

1  /*
2  -----
3  Laboratoire : 04
4  Fichier     : AbstactContainer.h
5  Auteur(s)    : Gabrielli Alexandre , Povoà Tiago
6  Date        : 22.05.2019
7
8  But          : Déclaration des méthode de la classe AbstactContainer. Cette classe abstraite
9  stoque une list de personne et permet de vérifier des règles elle demande à ces sous-class
10 d'offrir un affichage avec la méthode toString.
11 -----
12 */
13
14 #ifndef POO2_LABO4_CONTAINER_H
15 #define POO2_LABO4_CONTAINER_H
16
17 #include <string>
18 #include <list>
19 #include <memory>
20 #include <iostream>
21 #include "../Person/Person.h"
22
23 class AbstractContainer {
24
25 public:
26
27     /**
28      * Constructeur de la classe Container
29      * @param name nom du container
30      */
31     explicit AbstractContainer(const std::string &name);
32
33     /**
34      * verifie si l'état actuel correspond au regle
35      */
36     virtual bool verifie();
37
38     /**
39      * retire une personne du container
40      * @param person
41      * @return si la personne était dans le container
42      */
43     bool removeMember(const Person *person);
44
45     /**
46      * ajoute une personne au container
47      * @param person à ajouter
48      */
49     void addMember(Person *person);
50
51     /**
52      * Methode permettant de récupérer la liste de person
53      * @return liste de personne
54      */
55     const std::list<Person *> &getPersons() const;
56
57     /**
58      * Methode permettant de récupérer le nom du container
59      * @return nom du container
60      */
61     const std::string &getName() const;
62
63     friend std::ostream &operator<<(std::ostream &os, const AbstractContainer &container);
64
65 protected:
66     std::list<Person *> persons;
67
68     virtual std::string toString() const = 0;
69
70     std::string name;
71 };
72

```

```
73
74 #endif //POO2_LABO4_CONTAINER_H
75
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Bank.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2019  
7  
8 But : Déclaration des méthode de la classe Bank. La class Bank hérite de  
9 AbstractContainer et peu en plus être construit depuis une initializer_list de personne  
10 -----  
11 */  
12  
13 #ifndef POO2_LABO4_BANK_H  
14 #define POO2_LABO4_BANK_H  
15  
16  
17 #include <iostream>  
18 #include "AbstractContainer.h"  
19  
20 class Bank : public AbstractContainer {  
21  
22 public:  
23     /**  
24     * constructeur  
25     * @param name nom du conteneur  
26     */  
27     explicit Bank(const std::string &name);  
28  
29     /**  
30     * on peu aussi le construire depuis une liste de personne  
31     * @param name nom du conteneur  
32     * @param persons liste de personne  
33     */  
34     Bank(const std::string &name, std::initializer_list<Person *> persons);  
35  
36     /**  
37     * offre un affichage  
38     * @return une string  
39     */  
40     std::string toString() const override;  
41  
42 };  
43  
44  
45 #endif //POO2_LABO4_BANK_H  
46
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Boat.cpp  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2019  
7  
8 But : implémente les méthodes de la classe Boat.  
9 -----  
10 */  
11  
12 #include <iostream>  
13 #include "Boat.h"  
14  
15  
16 std::string Boat::toString() const {  
17     std::string string;  
18     string = this->name + " : <" ;  
19     for (const auto &person : this->persons) {  
20         string += person->getName() + " ";  
21     }  
22     string += " >";  
23  
24     return string;  
25 }  
26  
27 Boat::Boat(const std::string &string) : AbstractContainer(string) {}  
28  
29 bool Boat::verifie() {  
30     if (persons.size() > MAXBOATCAPACITY) {  
31         std::cout << "capacity max du bateau " << MAXBOATCAPACITY << std::endl;  
32         return false;  
33     }  
34     return AbstractContainer::verifie();  
35 }  
36  
37  
38 bool Boat::hasDriver() {  
39  
40     for (const auto &person : persons) {  
41         if (person->canDrive()) {  
42             return true;  
43         }  
44     }  
45     std::cout << "personne n'est en mesure de conduire" << std::endl;  
46     return false;  
47 }  
48  
49  
50
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : Boat.h  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2019  
7  
8 But : Déclaration des méthode de la classe Boat. La class Boat hérite de  
9 AbstractContainer  
10 -----  
11 */  
12  
13 #ifndef POO2_LABO4_BOAT_H  
14 #define POO2_LABO4_BOAT_H  
15  
16  
17 #include <iostream>  
18 #include "AbstractContainer.h"  
19  
20 #define MAXBOATCAPACITY 2  
21  
22 class Boat : public AbstractContainer {  
23  
24  
25 public:  
26     /**  
27     * constructeur  
28     */  
29     explicit Boat(const std::string &string);  
30  
31     /**  
32     * vérifie si l'état actuel correspond au règle (vérifie qu'il n'y ai pas plus de  
33     */  
34     bool verifie() override;  
35  
36     /**  
37     * offre un affichage  
38     * @return une string correspondant à l'affichage du container  
39     */  
40     std::string toString() const override;  
41  
42     /**  
43     * vérifie si l'une des personnes au moins à le droit de conduire  
44     */  
45     bool hasDriver();  
46  
47  
48 };  
49  
50  
51 #endif //POO2_LABO4_BOAT_H  
52
```

```

1  /*
2  -----
3  Laboratoire : 04
4  Fichier      : Controller.cpp
5  Auteur(s)    : Gabrielli Alexandre , Povoà Tiago
6  Date        : 22.05.2018
7
8  But          : définition de nos fonctions
9  -----
10 */
11
12 #include <Person/Papa.h>
13 #include <iostream>
14 #include <iomanip>
15 #include "Controller.h"
16
17 #define AFFICHERTOUCH 'p'
18 #define EMBARQUERTOUCH 'e'
19 #define DEBARQUERTOUCH 'd'
20 #define REINISIALISETOUCH 'r'
21 #define QUITTOUCH 'q'
22 #define MENUTOUCH 'h'
23 #define MOUVETOUCH 'm'
24
25 #define TUTOP "p : afficher\n"
26 #define TUTOE "e <nom>: embarquer <nom>\n"
27 #define TUTOD "d <nom>: debarquer <nom>\n"
28 #define TUTOM "m : deplacer bateau\n"
29 #define TUTOR "r : reinitialiser\n"
30 #define TUTOQ "q : quitter\n"
31 #define TUTOH "h : menu\n"
32 #define INPUTERROR "je n'est pas compris ce que vous voulez faire, appuyer sur h afficher l'aide"
33 #define NONAMEFOUND "ce nom ne semble pas correct"
34 #define NOPERSONINSID "la personne n'est pas dans "
35
36
37 #define LIGNESEPARATOR '-'
38 #define CENTERSEPARATOR '='
39 #define LINESIZE 60
40
41
42 Controller::Controller(std::initializer_list<Person *> persons) : turn(0), boatpositionisleft(true) {
43
44     for (Person *person : persons) {
45         this->persons.insert(name_person(person->getName(), person));
46     }
47     gauche = new Bank(RIVEGAUCHE, persons);
48     droite = new Bank(RIVEDROITE);
49     boat = new Boat(BOAT);
50
51     showMenu();
52     display();
53
54
55 }
56
57
58 void Controller::nextTurn() {
59     bool again = false;
60     turn++;
61     std::cout << turn;
62     /*pas de vérification de saisie utilisateur => pas de bufferOverflow*/
63     std::string choice;
64     std::getline(std::cin, choice);
65     std::cin.clear();
66     std::fflush(stdin);
67
68     switch (choice[0]) {
69         case AFFICHERTOUCH:
70             showMenu();
71             break;
72         case EMBARQUERTOUCH:

```

```

73         embark(choice.substr(2));
74         break;
75     case DEBARQUERTOUCH:
76         debark(choice.substr(2));
77         break;
78     case MOUVETOUCH:
79         if (boat->hasDriver())
80             boootpositionisleft = !boootpositionisleft;
81         break;
82     case REINISIALISETOUCH:
83         reset();
84         break;
85     case QUITTOUCH:
86         again = true;
87     case MENUTOUCH:
88         showMenu();
89         break;
90     default:
91         std::cout << INPUTERROR;
92         break;
93     }
94 }
95 display();
96 //ne quitte pas t'en qu'on a pas appuyer sur q
97 if (!again)
98     nextTurn();
99 }

100 void separatorLine(char separator) {
101     for (size_t i = 0; i < LIGNESIZE; i++) {
102         std::cout << separator;
103     }
104     std::cout << std::endl;
105 }
106 }

107

108 void Controller::showMenu() {
109     std::cout << TUTOP << TUTOE << TUTOD << TUTOM << TUTOR << TUTOQ << TUTOH << std::endl;
110 }

111

112 void Controller::display() {
113     separatorLine(LIGNESEPARATOR);
114     std::cout << *gauche << std::endl;
115     separatorLine(LIGNESEPARATOR);
116     boootpositionisleft ? std::cout << *boat << std::endl : std::cout << std::endl;
117     separatorLine(CENTERSEPARATOR);
118     !boootpositionisleft ? std::cout << *boat << std::endl : std::cout << std::endl;
119     separatorLine(LIGNESEPARATOR);
120     std::cout << *droite << std::endl;
121     separatorLine(LIGNESEPARATOR);
122 }

123

124

125 bool Controller::movePeople(const std::string &person, bool debark) {
126     AbstractContainer *containers[2];
127     auto it = persons.find(person);
128     if (it != persons.end()) {
129
130         Person *p = it->second;
131
132         containers[debark ? 1 : 0] = boootpositionisleft ? gauche : droite;
133
134         containers[debark ? 0 : 1] = boat;
135         if (containers[0]->removeMember(p)) {
136             containers[1]->addMember(p);
137         } else {
138             std::cout << NOPERSONINSID << containers[0]->getName() << std::endl;
139             return false;
140         }
141
142         if (!(containers[0]->verifie() && containers[1]->verifie())) {
143             containers[1]->removeMember(p);
144             containers[0]->addMember(p);

```

```
145         }
146     } else {
147         std::cout << NONAMEFOUND << std::endl;
148     }
149 }
150
151
152 bool Controller::debark(const std::string &person) {
153     return movePeople(person, true);
154 }
155
156 bool Controller::embark(const std::string &person) {
157     return movePeople(person, false);
158 }
159
160 void Controller::reset() {
161     turn = 0;
162
163     for (auto p : droite->getPersons()) {
164         gauche->addMember(p);
165         droite->removeMember(p);
166     }
167     for (auto p : boat->getPersons()) {
168         gauche->addMember(p);
169         boat->removeMember(p);
170     }
171
172     boooatpositionisleft = true;
173 }
174
175 Controller::~Controller() {
176     delete boat;
177     delete gauche;
178     delete droite;
179 }
```

```

1  /*
2  -----
3  Laboratoire : 04
4  Fichier     : Controller.h
5  Auteur(s)   : Gabrielli Alexandre , Povoà Tiago
6  Date        : 22.05.2018
7
8  But         : déclaration des méthodes de la classe Controller.
9  Cette classe gère le control de notre application, lorsque le 1er tour est relancer
10 elle relance automatiquement les tours suivants jusqu'à ce que l'utilisateur décide
11 de quitter le programme. Il n'y a pas de condition de victoire
12 -----
13 */
14
15 #ifndef POO2_LABO4_CONTROLLER_H
16 #define POO2_LABO4_CONTROLLER_H
17
18 #include <list>
19 #include <memory>
20
21 #include <Container/Bank.h>
22 #include <Container/Boat.h>
23
24 #define RIVEDROITE "droite"
25 #define RIVEGAUCHE "gauche"
26 #define BOAT "bateau"
27
28
29 class Controller {
30     typedef std::pair<std::string, Person *> name_person;
31 private:
32     int turn;
33     Bank *gauche;
34     Bank *droite;
35     Boat *boat;
36     bool boatpositionisleft;
37     std::map<std::string, Person *> persons;
38
39
40 public:
41     /**
42      * Constructeur de la classe Controller
43      * @param persons contiens les personnes avec lequel nous désirons jouer
44      */
45     Controller(std::initializer_list<Person *> persons);
46
47     /**
48      * Destructeur de la classe Controller
49      */
50     ~Controller();
51
52     /**
53      * joue un tour et relance un nouveau automatiquement à la fin du tour
54      */
55     void nextTurn();
56
57 private:
58     /**
59      * embarque une personne
60      * @param person nom de la personne a embarquer
61      * @return si on a pu réaliser l'action
62      */
63     bool embark(const std::string &person);
64
65     /**
66      * debarque une personne
67      * @param person nom de la personne a debarquer
68      * @return si on a pu réaliser l'action
69      */
70     bool debark(const std::string &person);
71
72     /**

```

```
73     * cette classe fait embarque/debark une personne d'une rive au bateau
74     * @param person le nom de la personne que l'on désire
75     * @param debark si true on débarque si false on embarque
76     * @return si on a pu réaliser l'action
77 */
78 bool movePeople(const std::string &person, bool debark);
79
80 /**
81     * cette méthode réinitialise le jeu
82     */
83 void reset();
84
85 /**
86     * Méthode qui affiche le menu du programme
87     */
88 void showMenu();
89
90 /**
91     * Méthode qui affiche le bateau et les deux rives
92     */
93 void display();
94 };
95
96
97 #endif //POO2_LABO4_CONTROLLER_H
98
```

```
1 /*  
2 -----  
3 Laboratoire : 04  
4 Fichier : main.cpp  
5 Auteur(s) : Gabrielli Alexandre , Povoà Tiago  
6 Date : 22.05.2018  
7  
8 But : le main initialise les personnes avec lequel l'application devra "jouer"  
9 elle instancie ensuite le contrôleur avec cette list et lance le premier tour  
10 -----  
11 */  
12  
13  
14 #include <Person/Voleur.h>  
15 #include <Person/Policier.h>  
16 #include <Person/Fille.h>  
17 #include <Person/Fils.h>  
18 #include <Person/Mamam.h>  
19 #include <Person/Papa.h>  
20 #include <iostream>  
21  
22 #include "Controller.h"  
23  
24 int main() {  
25     std::initializer_list<Person *> toBank = {  
26         new Papa("papa"),  
27         new Mamam("mamam"),  
28         new Fils("paul"),  
29         new Fils("pierre"),  
30         new Fille("julie"),  
31         new Fille("janne"),  
32         new Policier("policier"),  
33         new Voleur("voleur")  
34     };  
35  
36     Controller controller(toBank);  
37  
38     controller.nextTurn();  
39  
40     std::cout << "l'application va se terminer" << std::endl;  
41     return 0;  
42 }
```